-Perform the following hex computations (leave the result in hex):

a) 1234 +9876
b) 0FFF - 0F34
c) 100 - 1
d) 0FFE – 1

==========

How many hexadecimal digits in:

a) Byte b) word c) double word

==========

- Assuming a 16-bit two's complement format, determine which of the values below are positive and which are negative.

a) 0ABCD b) 1024 c) 0DEAD d) 0ADD e) 0BEEF
f) 8 g) 05AAF h) 0FFFF i) 0ACDB j) 0CDBA
k) 0FEBA l) 35 m) 0BA n) 0ABA o) 0BAD
p) 0DAB q) 4321 r) 334 s) 45 t) 0E65 u) 0BEAD
v) 0ABE w) 0DEAF x) 0DAD y) 9876

==========

- What three components make up Von Neumann Machines?

==========

- What is the purpose of

a) The system bus
b) The address bus
c) The data bus
d) The control bus

==========

6- Explain how to store a word in byte addressable memory (that is, at what addresses).?

==========

7- How many different I/O locations can you address on the 80x86 chip? How many are typically available on a PC?

==========

-Convert the following logical addresses to physical addresses. Assume all values are hexadecimal and real mode operation on the 80x86:

a) 1000:1000 b) 1234:5678 c) 0:1000 d) 100:9000 e) FF00:1000
f) 800:8000 g) 8000:800 h) 234:9843 i) 1111:FFFF j) FFFF:10

==========

The CPU can access operands (data) in various ways, called addressing modes. In 80x86 there are 7addressing modes , list these modes with an example of each one?

1. register MOV ES,AX
2. immediate ADD AL,40H
3. direct MOV [352F],AH
4. register indirect MOV [DI],AH
5. based relative MOV AL,[BP]+5
6. indexed relative MOV CL,[DI]+20
7. based indexed relative MOV AH,[BP][DI]+12

==========

-Describe a common use for each of the following addressing modes:

a) Register b) Displacement only c) Immediate
d) Register Indirect e) Indexed f) Based indexed
g) Based indexed plus displacement

==========

- List all of the 80x86 eight bit registers.
- List all the 80x86 general purpose 16 bit registers.
-List all the 80x86 segment registers (those available on all processors).

- Describe the "special purposes" of each of the general purpose registers.

==========

- What values appear in the 8086 flags register?

==========

In what segment (8086) would you normally place your variables?

==========

Consider CS = 2000h ; DS = 1500h ; DI = 0100h ; BX = 0130h ;
SS = 5000h ; SP = 0250h ; BP = 1400h ; AX = 4C69h ;
DX = 8855h ; CX= 6632h

(i) Find a physical address of the lower and upper range of CS, SS and DS memory .

Cs : lower address : 20000h upper address : 2FFFFh
SS : lower address : 50000h upper address : 5FFFFh
DS : lower address : 15000h upper address : 24FFFh

(ii) Find the logical and physical address of a data stored at offset address BX .

Logical address : 1500:0130 physical address : 15130h

(iii) Find the logical and physical address of a data stored at offset address BP.

Logical address : 5000:1400 physical address : 51400h

(iv) Find the contents of destination and the physical address accessed by each of the following instructions:

- MOV [Bp+100], AX
  Physical address : 51500h , contain 4C69h
- MOV SS: 5[BX][DI], DX
  physical address : 50235h , contain 8855h
- MOV [0200h], CL
  Physical address : 15200h , contain 32

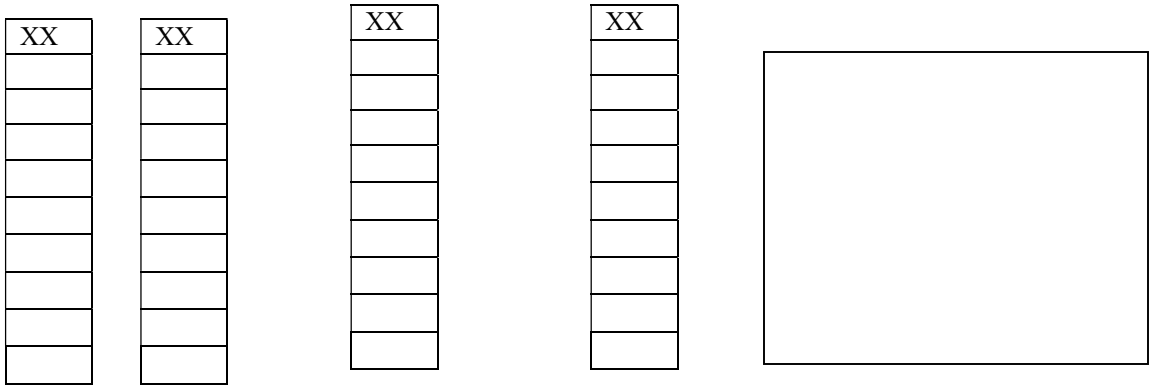
What are The addressing mode have used in each of the above instruction.

based relative
based indexed relative
register indirect

(v) What do the change in contents of the stack segment and stack segment register after the execution of the following instructions:

PUSH BX
PUSH DX
POP AX

No change will occurs at stack segment register (SS reg)



==========

* remember

Segment register	CS	DS	ES	SS
Offset register(s)	IP	SI, DI, BX	SI, DI, BX	SP, BP

Segment Override:

MOV AL,[BX]	DS:BX	however	MOV AL,ES:[BX]
MOV AX,[BP]	SS:BP	however	MOV AX,DS:[BP]

==========

-Provide an example that shows that it requires n+1 bits to hold the sum of two n-bit binary values.

ADC and SBB can be forced to behave exactly like ADD and SUB by inserting some other instruction before ADC and SBB. What instruction must be inserted in front of ADC to make it behave like ADD? In front of SBB to make it behave like SUB?

==========

- Provide four different ways to add two to the value in the BX register. No way should require more than two instructions?

==========

- Explain the difference between the carry flag and the overflow flag?

==========

- What is the difference between a "MOV reg, immediate" instruction and a "LEA reg , address" instruction?

==========

- Assume the following register contents: AX = 01FA BX = FF03

What is the NEW value of AX if the instruction " imul bl" is executed?

*AX = signed AL * signed BL = FA * 03 = -6 * +3 = -18, AX = FFEEh*

==========

- Assume the following register contents: AX = 001A BX = FFFC

What is the new value of AX if the instruction "idiv bl" is executed?

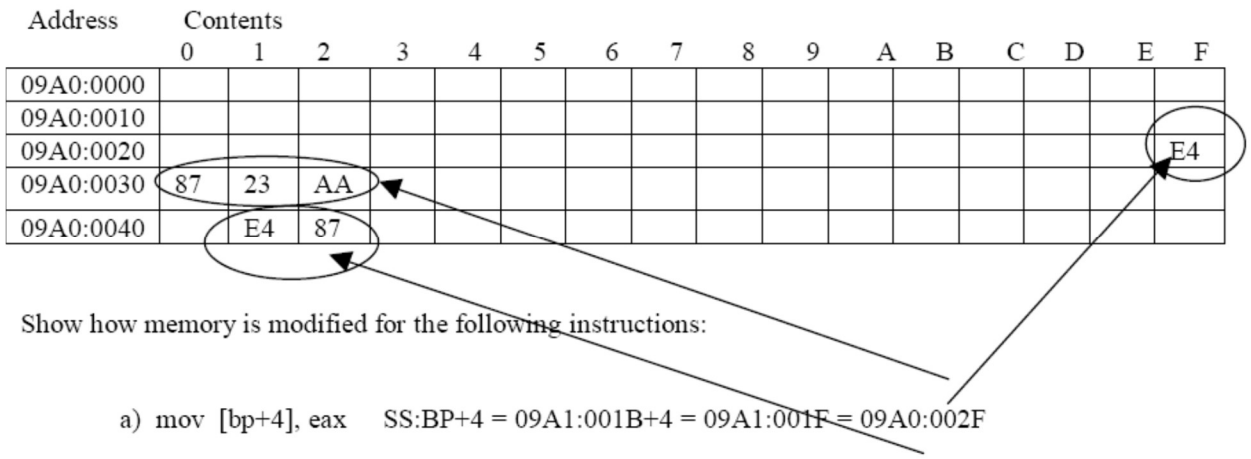
Remember that AL gets the quotient, and that AH gets the remainder.

Signed 001A / signed FC = 26/ -4, quotient(AL) = -6 = FA h, rmdr (AH) = +2 = 02 h

==========

- Assume the following register contents:

DS: 09A4, SS: 09A1, BX= 0005, BP:001B, EAX = AA2387E4



Show how memory is modified for the following instructions:

a) mov [bp+4], eax SS:BP+4 = 09A1:001B+4 = 09A1:001F = 09A0:002F

==========

- For each of the following sums, indicate if unsigned overflow (carry flag), signed overflow (two's complement) occur (neither, one or the other, or both)

a. F0h + FFh

NEITHER **Only Unsigned overflow** Only Signed Overflow Both

F0 + FF = FDh, sets carry flag. As signed: Negative + Negative = Negative, no signed overflow.

b. 90h + A0h

NEITHER Only Unsigned overflow Only Signed Overflow **Both**

90h + A0h = 30h, sets carry flag, As signed: negative + negative = positive, so signed overflow.

c. 10h + A0h

NEITHER Only Unsigned overflow Only Signed Overflow Both

10h + A0h = B0h, carry flag not set, as signed: postitive + negative cannot overflow.

d. 50h + 40h

NEITHER Only Unsigned overflow **Only Signed Overflow** Both

50h + 40h = 90h. carry flag not set, as signed: positive + positive = negative, so signed overflow.

e. 20h + 30h

NEITHER Only Unsigned overflow Only Signed Overflow Both

20h + 30h = 50h. Carry flag not set, as signed: positive + positive = positive, so no signed overflow.

==========

- The instruction "add ax, bx" will do a 16 bit add of AX = AX + BX. How could I do a 16 bit add if I only have 8 bit registers? Write a two instruction sequence that will do a 16 add of AX = AX+BX but you can ONLY use registers AH, AL, BH, BL in your instructions.

```
add al,bl
adc ah, bh          ;   ADD with CARRY (AH + BH + Carry Flag)
```

==========

- We talked about the MASM assembler in class.

a. What does the "DB" assembler instruction stand *for and what does it do?*

Define Byte, reserves 1 byte of memory storage

b. What the "DW" assembler instruction stand for and what does it do?

Define Word, reserves 1 word of memory storage

c. I mentioned that the first two instructions of ANY program should be something like:

```
START:    mov ax, @data
          mov ds, ax
```

WHY IS THIS NECESSARY?

The DATA SEGMENT must be initialized to point to your data - this is not done automatically. Most instructions that reference memory will use the data segment as the default segment register and you must initialize this to point to your data segment -- it is NOT done automatically by DOS.

==========

- The ASCII codes 4Dh, 53h, 55h represent what?

M S U

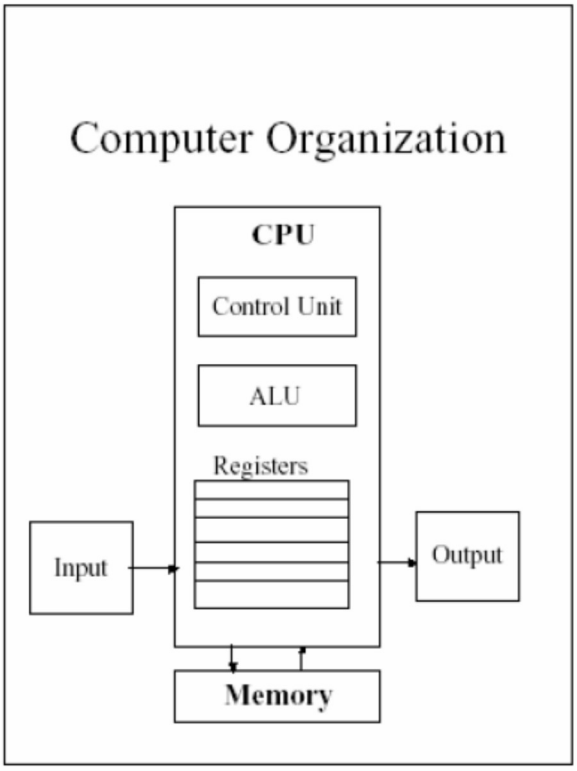
==========

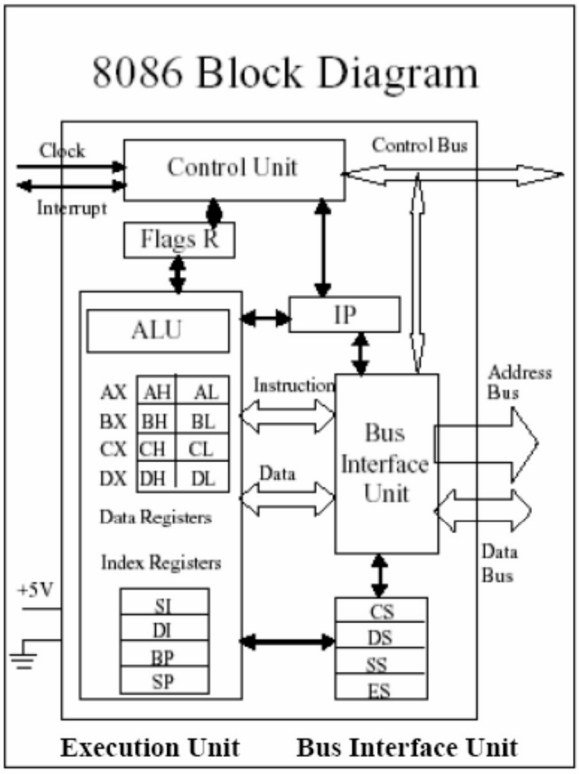
- Explain the difference between the following two instructions:

a. mov bx, 0200h b. mov bx, [0200h]

a. *Will move the value 0200h into BX*
b. *Moves the word stored at memory location 0200h into BX.*

- draw a block diagram of computer organization and 8086 INTERNAL ORGANIZATION





==========

- which of the following instruction is correct which is not :

MOV BX,14AFH ;	move 14AFH into BX	(correct)
MOV SI,2345H ;	move 2345H into SI	(correct)
MOV DI,2233H ;	move 2233H into DI	(correct)
MOV CS,2A3FH ;	move 2A3FH into CS	(incorrect)
MOV DS,CS ;	move the content of CS into DS	(correct)
MOV FR,BX ;	move the content of BX into FR	(incorrect)
MOV DS,14AFH ;	move 14AFH into DS	(incorrect)
MOV AL,123H ;		(incorrect)
MOV AX,3AFF21H ;		(incorrect)

==========

Remember :

Values cannot be loaded directly into (CS,DS,SS and ES)

MOV AX,1234H ; load 1234H into AX

MOV SS,AX ; load the value in AX into SS

==========

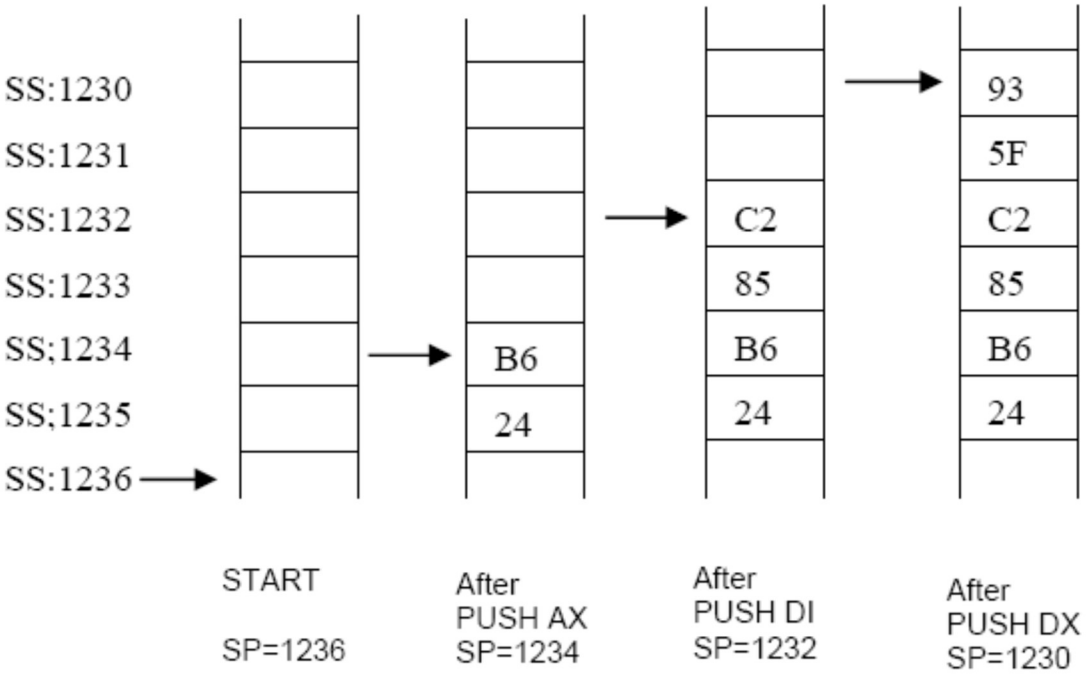
- using the inforamtion below, show the changes of the memory content (stack segment) after excute each instruction of the following:

SP=1236, AX=24B6, DI=85C2, and DX=5F93,

PUSH AX

PUSH DI

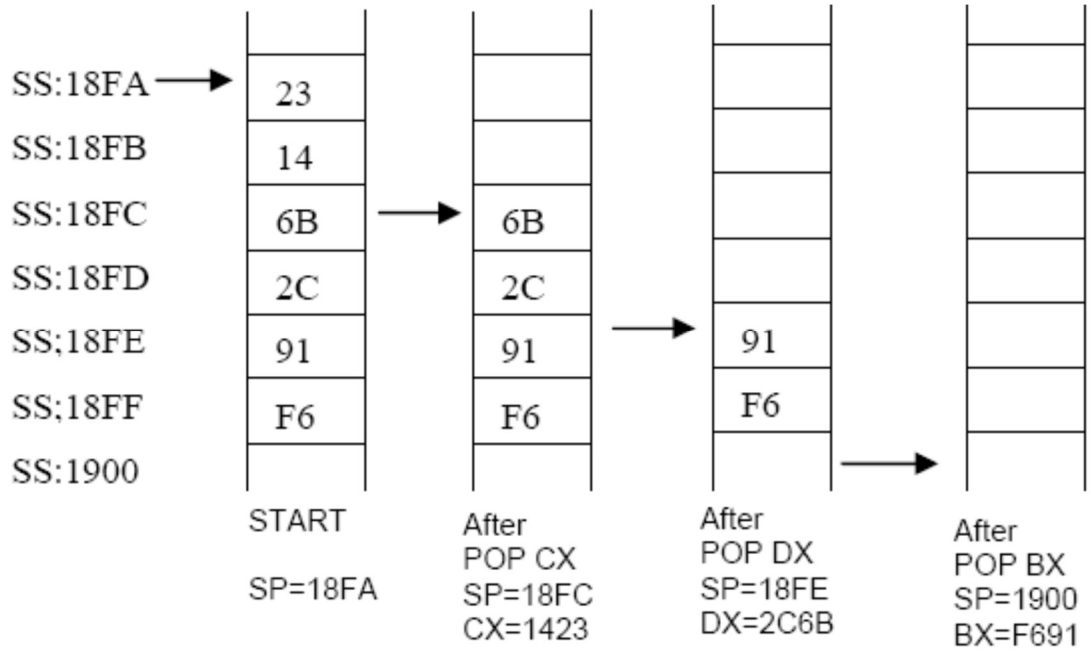
PUSH DX



==========

- assume that the stack is shown below, and SP=18FA, show the contents of the stack and registers as each of the following instructions is executed.

```
POP CX
POP DX
POP BX
```



==========

- Show how the flag register is affected by the addition of 38H and 2FH.

```
 MOV BH,38H          ;BH=38H
ADD BH,2FH           ;BH = BH + 2F = 38 + 2F= 67H
```

38	0011 1000
+ 2F	0010 1111
67	0110 0111

CF = 0 since there is no carry
PF = 0 since there is odd number of 1`s in the result
AF = 1 since there is a carry from d3 to d4
ZF = 0 since the result is not zero
SF = 0 since d7 of the result is zero.

==========

- Show how the flag register is affected by the following addition

```
MOV AX,34F5H      ;AX =34F5H
ADD AX,95EBH      ;AX = CAE0H
```

34F5	0011 0100 1111 0101
+ 95EB	1001 0101 1110 1011
CAE0	1100 1010 1110 0000

CF = 0 since there is no carry
PF = 0 since there is odd number of 1s
AF = 1 since there is a carry from d3 to d4
ZF = 0 since the result is not zero
SF = 1 since d15 of the result is 1.
 Note that the MOV instructions have no effect on the flag.

==========

```
; This program adds 5 unsigned byte numbers.
.MODEL SMALL
.STACK 64
.DATA
COUNT       EQU   05
DATA        DB    125,235,197,91,48
            ORG   0008H
SUM         DW    ?
.CODE
MAIN:       MOV AX, @DATA
            MOV   DS,AX
            MOV   CX,COUNT          ;CX is the loop counter
            MOV   SI,OFFSET DATA    ;SI is the data pointer
            MOV   AX,00             ;AX will hold the sum
BACK:       ADD   AL,[SI]           ;add the next byte to AL
            ADC   AH,00             ;add 1 to AH if CF =1
            INC   SI                ;increment data pointer
            DEC   CX                ;decrement loop counter
            JNZ   BACK              ;if not finished, go add next byte
            MOV   SUM,AX            ;store sum
            MOV   AH,4CH
            INT   21H               ;go back to DOS
            END   MAIN
```

==========

```
; This program is an example for Multiword addition
.MODEL SMALL
.STACK 64
.DATA
DATA1       DQ    548FB9963CE7H
            ORG   0010H
DATA2       DQ    3FCD4FA23B8DH
            ORG   00020H
DATA3       DQ    (?)
.CODE
MAIN:       MOV AX, @DATA
            MOV   DS,AX
            CLC                          ;clear carry before the first addition
            MOV   SI,OFFSET DATA1        ;SI is the data pointer for operand1
            MOV   DI,OFFSET DATA2        ;DI is the data pointer for operand2
            MOV   BX,OFFSET DATA3        ;BX is the data pointer for the sum
            MOV   CX,04                  ;CX is the loop counter
BACK:       MOV   AX,[SI]          ;move the first operand to AX
            ADC   AX,[DI]          ;add the second operand to AX
            MOV [BX],AX                  ;store the sum
            INC   SI                     ;point to next word of operand1
            INC   SI
            INC   DI                     ;point to next word of operand2
            INC   DI
            INC   BX                     ;point to next word of sum
            INC   BX
            LOOP  BACK                   ;if not finished, continue adding
            MOV   AH,4CH
            INT   21H                    ;go back to DOS
            END   MAIN
```

-Analyze the following program:

```
DATA_A DD 62562FAH
DATA_B DD 412963BH
RESULT DD ?
….. …..
MOV AX,WORD PTR DATA_A                ;AX=62FA
SUB AX,WORD PTR DATA_B                ;AX=AX – 963B
MOV WORD PTR RESULT,AX                ;save the result
MOV AX,WORD PTR DATA_A +2             ;AX=0625
SBB AX,WORD PTR DATA_B +2             ;SUB 0412 with borrow
MOV WORD PTR RESULT +2,AX             ;save the result
```

Note: PTR (Pointer) Directive is used to specify the size of the operand. Among the options for size are BYTE, WORD, DWORD and QWORD.

==========

- Assume the following memory contents at the start of each of the following instructions

Address	Contents															
09A0:0000	C5	67	A5	00	12	BC	34	BB	F4	72	09	A3	29	01	D4	CE
09A0:0010	FE	89	02	D8	A4	8A	7C	DD	90	3C	9B	83	65	19	F6	8A
09A0:0020	A7	CC	9A	BD	8E	90	2C	59	1C	90	0E	13	8C	39	58	C6
09A0:0030	76	D7	CA	FF	D8	71	18	24	40	A8	2C	76	93	C5	0F	9E
09A0:0040	82	A6	54	2E	9A	20	0A	98	E4	A0	0E	25	38	29	2C	86

65 E2

Assume the following register contents at the START of each of the following instructions.
ES: 09A0, DS: 09A0, SS: 09A1
AX = E265, DX = 73A2, CX = 0000, SP = 0018, BP=002E
Give the value of the affected register OR affected memory location after each instruction.
IF MEMORY IS MODIFIED, you MUST show the modified locations on the ABOVE memory map! LIST ALL registers that are affected by the instructions except for the flag registers.

a. Push ax ,sp = 0016, see map above (SS:SP = 09A1:0018 = 09A0: 0028)
b. sar dl , 2 dl=E8h
c. shr dl,2 dl= 28h
d. shl al,1 al = CAh
e. xor dl, al dl = C7h
f. or dl, F0h dl = C7h
g. not dl dl = F2h
h. pop dx dx = 901C h sp=001A
i. mul dl ax = al * dl = 101 * 162 = 16362 = 3FEA h
j. imul dl ax = ax * dl = 101 * -94 = -9494 = DAEA h
k. and dl, 22 dl = 22 h

==========

- Assume the same register contents/memory contents as above. For EACH of the following two instruction sequences, tell if the jump is TAKEN or NOT TAKEN.

a. cmp al,dl
jne there TAKEN (al not equal to dl)

b. cmp al,dl
jl there NOT_TAKEN (al, a postive number, is not less than dl, a negative number)

c. cmp al,dl
ja there NOT_TAKEN (al is not higher than dl)

d. cmp ax,dx
jg there NOT_TAKEN (ax, a negative number is not greater than dx, a postive number)

e. cmp ax,dx
jb there NOT_TAKEN (ax is not below dx)

f. test al,1
jnz there TAKEN (result of al AND 1 is non-zero, so branch taken)

g. add al,dl
jnc there NOT_TAKEN (al+dl produces a carry, a branch not taken)

h. add al,dl
js there NOT_TAKEN (MSB of al+dl is '0', so branch not taken)

i. add al, 40h
jno there NOT_TAKEN (al + 40h produces an overflow, so branch not takend)

==========

- Register AL has an 8 bit value ($b_7b_6b_5b_4b_3b_2b_1b_0$). Use a single logical instruction to change the contents of AL to ($b_700000b_1b_0$). Bits B7, B1, B0 unchanged, other bits set to zero.

And al, 83h

==========

- Register AL has an 8 bit value ($b_7b_6b_5b_4b_3b_2b_1b_0$). Use a single logical instruction to change the contents of AL to ($1b_6b_5b_4b_311b_0$). Bits B7, B2,B1, set to '1', other bits unchanged.

```
Or al, 86h
```

==========

- Write a subroutine that will return the number of bytes in a string. The string is terminated by a ’0’ byte (count DOES NOT INCLUDE the ’0’ byte), and the starting address of the string is passed to the subroutine via the DS:SI register. The count should be returned as zero if the string is ’empty’ (first byte is zero). The count should be returned in the AL register (maximum number of characters will be 255).

```
Strlen      proc
            Xor ax,ax
     Lp1:   mov bl, [si]
            Cmp bl,0
            Jz   exit
            Inc al
            Inc si
           Jmp lp1
    Exit:  ret
Strlen      endp
```

==========

- Write a subroutine that will return the maximum 16-bit SIGNED integer from an array of integers. On subroutine entry, register SI will point to the start of the array (each element is 16 bits), and register CX will have the number of integers in the array. The maximum value should be returned in the AX register. An example call to this procedure is shown below:

```
.data
        Myarray      dw    -45, 1000, -34, 1500, 20, 60
       Count    equ 6                                   ; six elements in the array
.code
   mov ax,@data
  mov ds,ax

            mov cx, count
           lea si, [myarray]
          call findmax
           …..
           …..

            Findmax      proc
                                          ;;; will assume that array always has at least 1 element

                    mov ax, 8000h          ;; get most negative 16-bit value into ax
              lp1:  cmp ax, [si]
                    jge   skip
                   mov ax, [si]            ;; ax is less than [SI], get memory value
             skip:  lea si, [si+2]         ;; increment pointer by 2 bytes
                   loop lp1                ;; cx has count
                  ret
       findmax       endp
```

==========